Beroepsproduct: parser (B_Compiler)

Februari 2021, v1.2

1. Inleiding

Zoals je weet wordt de opmaak van webpagina's gespecificeerd in Cascading Style Sheets, oftewel CSS. Ontwerpkeuzes die gemaakt zijn in deze opmaaktaal maken het soms wat omslachtig om opmaak te beschrijven. Ook krijg je vaak herhalende code. Om deze problemen aan te pakken zijn er verschillende CSS preprocessors zoals SASS en LESS die een eigen CSS "dialect" vertalen naar standaard CSS-broncode. In deze opdracht ga je zelf een vergelijkbare preprocessor maken.

2. Opdrachtomschrijving

Je gaat in deze opdracht dus een eigen CSS-dialect maken: ICSS-20-SEP. Een informele beschrijving van deze taal is te vinden in appendix A. Lees deze beschrijving goed! Je bouwt in deze opdracht verder aan een interactieve Java-applicatie: de ICSSTool. Deze tool is een interactieve compiler. Je kunt er interactief ICSS in bewerken en deze stapsgewijs compileren naar CSS. Deze CSS kun je vervolgens exporteren. Het raamwerk voor de ICSS tool krijg je als startcode aangeleverd. De GUI is al gemaakt en alle onderdelen zijn in minimale vorm aanwezig. De opdracht bestaat uit het volledig maken van de tool door een serie deelopdrachten.

3. Inleveren

Met betrekking tot het inleveren gelden de volgende eisen:

- De opdracht dient in iSAS te worden ingeleverd. Kijk voor de exacte datum en tijd in iSAS.
- Alle onderdelen dienen gebundeld in één ZIP-bestand te worden aangeleverd.
- Bijlagen mogen alleen in PDF worden aangeleverd.
- Er moet een bijlage aanwezig zijn waarin je aangeeft (1) aan welke eisen het product voldoet, en (2) wat de met de docent overeengekomen taaluitbreiding is en hoeveel punten deze maximaal waard is.

4. Eisen

Door de deelopdrachten te realiseren kun je in totaal 120 punten verdienen. Je cijfer is het aantal punten gedeeld door 12.

Cesuur-regels:

- Als je niet aan alle "Must" eisen voldoet krijg je maximaal 60 punten (cijfer = 5). Let op, er zijn ook Musts die geen punten opleveren!

- Een voldoende haal je door minimaal alle Musts te implementeren. Het minimum aantal punten wat je dan behaalt is 70 punten (inclusief mogelijke kleine foutjes). Je cijfer is dan een 5,8

Alle Musts foutloos implementeren levert 90 punten op (7,5). Door de shoulds te implementeren en een eigen uitbreiding te maken kun je een 10 halen. We dagen je uit voor de 10 te gaan, dat is zeker te doen!

4.1 Algemene eisen (0 punten)

De code die je oplevert is een uitbreiding op de bij de opdracht beschikbaar gestelde startcode. Voor de code gelden de volgende algemene eisen.

ID	Omschrijving	Prio	Punten	Competentie VT
AL01	De code behoudt de packagestructuur van de aangeleverde startcode. Toegevoegde code bevindt zich in de relevante packages.	Must	0	APP-6
AL02	Alle code compileert en is te bouwen met Maven 3.6 of hoger, onder OpenJDK 13. Tip: controleer dit door **eerst** `mvn clean` uit te voeren alvorens te compileren en in te leveren, hierop een onvoldoende halen is echt zonde. **Gebruik van Oracle versies van Java is uitdrukkelijk niet toegestaan**.	Must	0	n.v.t.
AL03	De code is goed geformatteerd, zo nodig voorzien van commentaar, correcte variabelenamen gebruikt, bevat geen onnodig ingewikkelde constructies en is zo onderhoudbaar mogelijk opgesteld. (naar oordeel van docent)	Must	0	n.v.t.
AL04	De docent heeft vastgesteld (tijdens les, assessment of op een andere manier) dat de compiler eigen werk is en dat je voldoet aan de beoordelingscriteria van APP-6, te weten: - Kent de standaardarchitectuur van compilers; - Kent de basisbegrippen over programmeertalen (zoals syntaxis, semantiek).	Must	0	APP-6

4.2 Parseren (40 punten)

De ICSS tool moet uiteraard een parser component bevatten. Deze parser realiseer je door een ANTLR grammatica op te stellen voor de concrete syntax van ICSS. Vervolgens implementeer je een listener voor ANTLR die de parse tree omzet naar een AST. De abstracte syntax van de taal is al voor je gedefinieerd in de package `nl.han.ica.icss.ast`. De grammatica met voorgegeven lexer regels kun je vinden in de `ICSS.g4` file en de (lege) `ASTListener` in de package `nl.han.ica.icss.parser`.

Een paar belangrijke tips:

- bestudeer de AST voordat je met de parser aan de slag gaat; je zult zien dat de parser en de AST qua structuur overeenkomsten hebben. Als je die

inziet, is het construeren van de parser eenvoudiger
- Aan het project zijn (unit)tests toegevoegd. Met deze tests kun je controleren of jouw parser de verwachte AST opbouwt.
- Het is toegestaan (helper) methodes toe te voegen aan de AST zodat je efficiëntere code kunt schrijven. Het is niet toegestaan de AST structureel te wijzigen.

Per onderdeel PA kun je 0, 5 of 10 punten krijgen. **5 punten worden alleen toegekend bij een kleine (onvoorziene) fout (oordeel docent); het weglaten van een deel uit de taalbeschrijving levert een onvoldoende op**. Bij 5 punten geldt de Must wel als behaald. Let op dat de tests niet uitputtend zijn. Het slagen van de test wil dus niet per definitie zeggen dat je parser volledig is. Het geeft alleen aan dat het inputbestand level[x].icss goed is geparseerd. **Maak zelf meer inputbestanden om te testen**.

ID	Omschrijving	Prio	Punten	Competentie VT
PA00	De parser dient zinvol gebruik te maken van **jouw** eigen implementatie van een stack generic voor `ASTNode` (VT: zie huiswerk `IHANStack<ASTNode>`)	Must	0	APP-1, APP-9
PA01	Implementeer een grammatica plus listener die AST's kan maken voor ICSS documenten die "eenvoudige opmaak" kan parseren, zoals beschreven in de taalbeschrijving. In `level0.icss` vind je een voorbeeld van ICSS code die je moet kunnen parseren. `testParseLevel0()` slaagt.	Must	10	APP-6, APP-7
PA02	Breid je grammatica en listener uit zodat nu ook assignments van variabelen en het gebruik ervan geparseerd kunnen worden. In `level1.icss` vind je voorbeeldcode die je nu zou moeten kunnen parseren. `testParseLevel1()` slaagt.	Must	10	APP-6, APP-7
PA03	Breid je grammatica en listener uit zodat je nu ook optellen en aftrekken en vermenigvuldigen kunt parseren. In `level2.icss` vind je voorbeeld- code die je nu ook zou moeten kunnen parseren. Houd hierbij rekening met de rekenregels (vermenigvuldigen gaat voor optellen en aftrekken, optellen en aftrekken gaan van links naar rechts; zie ook deze site."`testParseLevel2()` slaagt.	Must	10	APP-6, APP-7
PA04	Breid je grammatica en listener uit zodat je if/else-statements aankunt. In `level3.icss` vind je voorbeeldcode die je nu ook zou moeten kunnen parseren. `testParseLevel3()` slaagt.	Must	10	APP-6, APP-7
PA05	PA01 t/m PA04 leveren minimaal 30 punten op	Must	0	nvt

4.3 Checken (30 punten)

Tijdens het compileren van ICSS naar CSS willen we eerst controleren of de code naast syntactisch correct, ook semantisch correct is. Dit doe je door de checker component (`nl.han.ica.icss.checker.Checker`) te implementeren. Als je fouten detecteert in de AST kun je deze in de knopen van de AST opslaan met de `setError` methode.

Per CH onderdeel kun je 0 of de aangegeven punten krijgen.

ID	Omschrijving	Prio	Punten	Competentie VT
CH00	Minimaal vier van onderstaande checks **moeten** zijn geïmplementeerd	Must	0	APP-2, APP-6, APP-7,
CH01	Controleer of er geen variabelen worden gebruikt die niet gedefinieerd zijn.	Should	5	
CH02	Controleer of de operanden van de operaties plus en min van gelijk type zijn. Je mag geen pixels bij percentages optellen bijvoorbeeld. Controleer dat bij vermenigvuldigen minimaal een operand een scalaire waarde is. Zo mag `20% * 3` en `4 * 5` wel, maar mag `2px * 3px` niet.	Should	5	
CH03	Controleer of er geen kleuren worden gebruikt in operaties (plus, min en keer).	Should	5	
CH04	Controleer of bij declaraties het type van de value klopt met de property. Declaraties zoals `width: #ff0000` of `color: 12px` zijn natuurlijk onzin.	Should	5	
CH05	Controleer of de conditie bij een if-statement van het type boolean is (zowel bij een variabele-referentie als een boolean literal)	Should	5	
CH06	Controleer of variabelen enkel binnen hun scope gebruikt worden	Must	5	

Als je deze deeleisen geïmplementeerd hebt, kan je nu ook controleren of de input ICSS semantisch klopt. Voor de volgende fases in de compiler kun je er dus van uit gaan dat je met een correcte AST en gevulde symboltable verder kunt werken.

4.4 Transformeren (20 punten)

Om het genereren van de code makkelijker te maken gaan we de AST in een aantal stappen vereenvoudigen. Hiervoor is een evaluatie-klasse gedefinieerd in `nl.han.ica.icss.transform` genaamd `Evaluator`.

LET OP!! De transformaties zijn afhankelijk van elkaar. Daarom moeten ze in een enkele tree-traversal uitgevoerd worden. Je kunt de gecombineerde transformatie testen met de volgende ICSS-code (bedenk zelf wat het resultaat is):

```
AdjustWidth := TRUE;
WidthVar := 0px;

p {
    if [AdjustWidth] {
        WidthVar := 200px;
    } else {
        WidthVar := 250px;
    }

    width: WidthVar;
}
```

Per TR onderdeel kun je 0, 5 of 10 punten krijgen. 5 punten krijg je alleen bij een kleine (onvoorziene) fout (ter beoordeling van docent). In dat geval geldt de "Must" als behaald.

ID	Omschrijving	Prio	Punten	Competenties
TR01	Evalueer expressies. Schrijf een transformatie in `Evaluator` die alle `Expression` knopen in de AST door een `Literal` knoop met de berekende waarde vervangt.	Must	10	APP-2, APP-6, APP-7
TR02	Evalueer if/else expressies. Schrijf een transformatie in `Evaluator` die alle `IfClause`s uit de AST verwijdert. Wanneer de conditie van de `IfClause` `TRUE` is wordt deze vervangen door de body van het if-statement. Als de conditie `FALSE` is dan vervang je de `IfClause` door de body van de `ElseClause`. Als er geen `ElseClause` is bij een negatieve conditie dan verwijder je de `IfClause` volledig uit de AST.	Must	10	APP-2, APP-6, APP-7

4.5 Genereren (10 punten)

De laatste stap is het genererereren van CSS2-compliant code vanuit ICSS code. Dit doe je door een tree- traversal op de volledig getransformeerde AST te doen om een string op te bouwen.

Per GE onderdeel kun je 0 of 5 punten krijgen.

ID	Omschrijving	Prio	Punten	Competentie VT
GE01	Implementeer de generator in `nl.han.ica.icss.generator.Generator` die de AST naar een CSS2-compliant string omzet.	Must	5	APP-2, APP-6, APP-7
GE02	Zorg dat de CSS met twee spaties inspringing per scopeniveau gegenereerd wordt.	Must	5	APP-2, APP-6, APP-7

4.6 Eigen uitbreidingen (20 punten)

Je mag een eigen taaluitbreiding bedenken en implementeren. Je spreekt met de docent af hoeveel extra punten je voor je idee kunt halen. Met deze eigen functionaliteiten kun je maximaal 20 punten verdienen. Denk bijvoorbeeld aan:

- Het implementeren van optimalisaties op de AST.
- Het implementeren van een for-lus (moeilijk!)
- Implementeren van booleaanse expressies zoals `3<5`, `Value==5`, `!AdjustWidth` etc.
- Iedere variabele mag alleen een vast type hebben. Dan mag `Var := 10px;` en daarna `Var := 5%;` niet voorkomen.

Appendix A ICSS-20-SEP: Informele Specificatie

ICSS is een opmaaktaal vergelijkbaar met Cascading Stylesheets (CSS). Het heeft niet alle mogelijkheden van CSS, maar tegelijkertijd heeft het ook een aantal features die CSS niet heeft. Dit document beschrijft informeel de ICSS-20-SEP versie van ICSS.

Eenvoudige opmaak

ICSS gebruikt net als CSS *rules* om de opmaak van HTML elements aan te geven. Een stylesheet bestaat uit een aantal regels die na elkaar worden toegepast op een HTML document. Regels hebben de vorm `<selector> { <declaraties> }`. Hierin is de selector ofwel een specifiek type tag die geselecteerd kan worden, ofwel een element met een unieke ID, ofwel elementen van een bepaalde class. Elementen met een uniek ID worden aangegeven door een identifier beginnend met een hekje (`#`) en elementen in een class worden aangegeven door de klassenamen voorafgegaan door een punt (`.`). Declaraties zijn naam/waarde paren van de vorm `<property>: <value>;`. Sommige waardes kunnen ook een eenheid bevatten zoals `px` of `%`. Hier volgt een eenvoudige ICSS-stylesheet die tevens een CSS-stylesheet is:

```
a {
  color: #ff0000;
```

```
  background-color: #eeeeee;
}
#menu {
  width: 100%;
  height: 50px;
}
.active {
  color: #00ffff;
}
```

Beperkingen

ICSS is beperkter dan CSS. Dit zijn de beperkingen:

- Selectoren zijn allemaal lowercase.
- Selectoren selecteren maar op één ding tegelijk. Combinaties zoals `a.active` zijn niet toegestaan.
- Selectoren voor het selecteren van kinderen uit CSS zoals `div > a` zijn niet toegestaan.
- Alleen de properties `color`, `background-color`, `width` en `height` zijn toegestaan.
- Voor kleuren (`color` en `background-color`) moet de waarde als een hexadecimale waarde van zes tekens opgegeven worden (bijv. `#00ff00`)
- Voor groottes mag of een waarde in pixels (bijv. `100px`) of een percentage (bijv. `50%`) gespecifieerd worden.

Variabelen

Een feature die CSS niet heeft, maar ICSS wel is de definitie van variabelen. In ICSS kun je expressies een naam geven en dan op meerdere plaatsen waar je anders een waarde zou invullen die naam gebruiken. Een assignment van een variable ziet er als volgt uit: `Myvar := 100px;` Het gebruik ervan is dan: `width: Myvar;` Je kunt natuurlijk ook variabelen de waarde geven van een eerder gedefinieerde variabele: `Textcolor := Bgcolor;`

Voor gebruik in if-expressies (zie verderop) is het ook mogelijk booleans aan variabelen toe te kennen. Een boolean heeft de waarde `TRUE` of `FALSE`. Voorbeelden:

```
UseColor := TRUE;
SetHeight := FALSE;
```

Variabelenamen beginnen altijd met een hoofdletter. Variabelen mogen worden direct in de body van een stylesheet, stylerule, of een if/else-statement gedefinieerd.

Scope-regels

- een variabele gedeclareerd op body-niveau van de stylesheet is in scope voor alle stylerules die daarna komen
- Een variabele refereren mag alleen wanneer deze in de regels voorafgaand aan de referentie is gedeclareerd, in dezelfde scope, of in een globalere scope.
- Als een assignment binnen een rule gebeurt, is de scope van die variabele beperkt tot regels na het assignment binnen die rule.
- Als een assignment binnen een if-statement gebeurt, is de scope van die variabele beperkt tot binnen die if-expressie (idem voor de else-clauses).

If-expressies (met else-clause)

If-expressies mogen alleen voorkomen binnen CSS-rules. Binnen if-expressies mogen andere if-expressies voorkomen. Een if-expressie heeft de volgende syntaxis: `if [<expression>] { <body> }` waarbij de body dezelfde elementen mag bevatten als een rule. De accolades zijn altijd verplicht (dat is dus anders dan in bijv. Java). Een else-clause kan optioneel worden toegevoegd. De syntaxis is dan: `if [<expression>] { <body> } else { <body> }`

Voorbeeld van een ICSS stylesheet met if-expressies:

```
LinkColor := #ff0000;
ParWidth := 500px;
AdjustColor := TRUE;
UseLinkColor := FALSE;

p {
    width: ParWidth;
    if[AdjustColor] {
        color: #124532;
        if[UseLinkColor]{
            background-color: LinkColor;
        } else {
            background-color: #000000;
        }
    }
    height: 20px;
}
```

Bovenstaand voorbeeld vertaald naar de volgende CSS 2:

```
p {
  width: 500px;
  color: #124532;
  background-color: #000000;
  height: 20px;
```

```
}
```

Berekende waardes

Een andere uitbreiding in ICSS is de mogelijkheid om eenvoudige berekeningen te doen met waardes. In ICSS mag je optellen, aftrekken en vermenigvuldigen. Dit mag zowel in CSS-declaraties als in assignments van variabelen.

Bijvoorbeeld:

```
div {
  width: 50px + 2 * 10px - 2px; // dit is dus 68px
}
#menu {
  height: 20px + MyHeight * 2;
}
```

of

```
Menusize := Headersize - 20%;
```

Je mag alleen pixelwaardes bij pixelwaardes optellen en percentages bij percentages. Kleuren kun je niet optellen. Vermenigvuldigen gaat met minstens 1 scalaire waarde als operand. Voorbeelden: 3*5px of 4% *18*2*3. Je mag enkel scalaire waarden natuurlijk niet gebruiken als waarde van properties, want dan hebben ze geen eenheid (bijv. `px`). Let op, de gebruikelijke rekenregels moeten gelden voor optellen en vermenigvuldigen. Vermenigvuldigen gaat boven optellen en aftrekken.